Error Warning Summarization Utility

An Application Guide

Table of Contents

Introduction

The Intel® IA-64 instruction set architecture provides a 64-bit address space, which expands the available application space and introduces new features that can be used to eliminate application performance bottlenecks. Along with the new 64-bit architecture comes a new application programming model, with data types to support the manipulation of 64-bit objects. Migrating existing application code to this new 64-bit programming model requires modifications to the code through a process sometimes referred to as *code cleaning*.

During the code cleaning process, the IA-64 architecture compiler and linker tools can help identify problem areas by issuing errors and warnings as these tools encounter 64-bit issues. Such errors and warnings can number in the thousands depending on the size of the application. Some warnings are innocuous, while others must be examined.

The Error Warning Summarization Utility provides a means of organizing and planning your code cleaning effort. This utility

- processes the log files from an application build and generates reports that summarize the error and warning output.
- provides summary statistics to use for tracking overall progress (and regression).
- provides increasingly detailed reports – even down to the file and line level – about where errors and warnings occur.
- filters existing IA-32 architecture warnings and writes reports directly to a Microsoft Excel* spreadsheet, if Excel is available.

The utility recognizes errors and warnings generated in the build process by a range of tools, including the compiler, the resource compiler, the make utility, the assembler, and the linker. The utility supports the Intel® C/C++ compiler and the Microsoft Visual C++* compiler and tools. Fortran compilers are not currently supported.

This application note describes the Error Warning Summarization Utility and its usage and design. Use this document and this utility to strategize your porting effort, to track your application's progress, and to watch for regressions as development continues.

Error Warning Summarization Utility

Usage

The Error Warning Summarization Utility takes application build-logs as input, processes the error and warning information, and produces summarizing reports. The build-log files can be generated in a number of ways. If you use the Microsoft Developer Studio* IDE environment, then the build-logs are written to files in the project directory with a .plg file extension.
If you build on the command line using makefiles, then you can redirect the makefile output to a file and use it as input.

```
# Redirect stderr and stdout to "logfile.txt"

prompt> nmake >logfile.txt 2>&1
```

The utility is written in the Python open source programming language(http://www.python.org) and is invoked from the command line. See the Appendix section for Python installation instructions. The utility invocation options are designed to support several different ways that applications are built.

- **Single Report:** This invocation takes one or more files listed on the command line and produces one report summarizing across all files. This works well for application builds that result in multiple `.plg` files.

```
prompt> python err_warn.py <list of logs on the command line>
```

or

```
prompt> python err_warn.py project1.plg subdir1\project2.plg
```

- **Single Report – Find:** This invocation removes the need to supply the log files as command line arguments. Instead, a basic pattern expression can be used to identify the log files. All files in the current directory and in subdirectories matching the pattern will be included in the final report. For example, specifying `*.plg` as the pattern will cause all .plg files in the current directory and in subdirectories to act as log file inputs. The quotation marks in the following examples are necessary for correct syntax.

```
prompt> python err_warn.py -find "<pattern>"
```

or

```
prompt> python err_warn.py -find "*.plg"
```

- **Multi-Report:** This invocation is intended for the scenario where build-logs are collected over time in a directory and you want to generate reports for each of them, one by one, using a single invocation of the utility.

```
prompt> python err_warn.py -multi <list of logs on the command line>
```

or

```
prompt> python err_warn.py -multi log_Mar1.txt log_Mar16.txt log_Apr1.txt
log_Apr16.txt
```

- **Filtering:** This invocation generates reports based on the first log file, less the errors and warnings found in the second log file.

```
prompt> python err_warn.py -filter <logfile1> <not logfile2>
```

or

```
prompt> python err_warn.py -filter log_IA64.txt log_IA32.txt
```

■ **Flags**

```
-help       # Display usage information
-version    # Show version information
-Excel      # Send reports directly to Microsoft Excel
-Intel      # Parse output from the Intel compiler
              (default is Microsoft compiler)
-Alltools   # Summarize all errors and warnings
              (default is compiler only)


Example:
prompt> python err_warn.py -Intel -Excel -Alltools -find "*.plg"
```

The filtering option specifically targets build environments that have existing IA-32 warnings. These warnings may have been examined already and may have been found to be innocuous. They may show up during the IA-64 build as well. By filtering them out, you can better focus on the new errors and warnings that require examination. To enable this filtering, the summarization utility provides the *-filter* option, which first processes the first log file and then makes a second pass, removing any warnings that are duplicates in the second log file. To be precise, duplicate warnings are defined as warnings that are present in both log files and that are associated with the same file name, line number, and error number.

If you have trouble with the filter option, consider keeping the compiler front-ends the same. For example, use only IA-32 and IA-64 compilers from Intel or use IA-32 and IA-64 compilers from Microsoft with this option. As different products, errors and warning reporting for Intel and Microsoft compilers are not necessarily the same.

The filter option should be used with care, since a non-problematic IA-32 warning may actually be very problematic for the 64-bit environment. At least one full inspection without filtering should be carried out at the end of the code cleaning process to verify that all remaining warnings do not cause problems in the 64-bit world.

The *-Excel* option uses the Microsoft Excel COM interface to create a workbook and populate its worksheets with data. It has been tested on Microsoft NT* 4.0 SP5, with Microsoft Office* 97 and with Microsoft Office 2000 and with the latest available Microsoft Office service packs. This option requires Mark Hammond's Pythonwin package, in addition to the standard Python distribution for Microsoft Windows*. It has been tested with Pythonwin versions 127 through 129. (See the Appendix for instructions on installing Pythonwin.)

The output file names depend on the method that you use to invoke the utility. Here is a table listing the names of the output files. In the case of the Multi-Report, the output file names correspond to the names of the input log files, regardless of whether you use the *-find* option to obtain the input logs or not. In the *"-multi -Excel -find <pattern>"* case, the spreadsheets will reside in the same directory as the log files that are found in the search.

Method	No -Excel flag	With -Excel flag
Single Report (and with *-find*)	stdout / console	File: ErrorWarnReport.xls
Multi-Report	File: <logfile1>.rpt File: <logfile2>.rpt File: ...	File: <logfile1>.xls File: <logfile2>.xls File: ...
Filtering	stdout / console	File: ErrorWarnReport.xls

Reports

The table below shows the list of the report formats that are generated by the utility. Each report format is tab-delimited and sorted by count. With the exception of output published to Microsoft Excel, each of these reports are joined into a master report, separated by a `"===="` identifier, before being written out. In the case of Microsoft Excel reports, each report is a separate worksheet within one master document. All report formats in the following table combine to make up one `.rpt` file or `.xls` output file described above.

Report Name	Format	Description
Summary	Total Files Compiled: Total Files With Errors/Warnings: Total Errors: Total Warnings: Total Unique Errors: Total Unique Warnings: Total Unique Errors or Warnings:	Provides top-level summary statistics.
Summary by Error / Warning Number	Table consisting of: (“error” or “warning”, errwarn_number, count)	Provides top-level information organized by error and warning number.
Summary by Directory	Table consisting of: (directory , count)	Sits at the top of the report hierarchy and shows directories that have the most errors or warnings.
Summary by File and Error / Warning Number	Table consisting of: (“error” or “warning”, filename , errwarn_number, count)	Shows the files with the most errors and warnings. It also takes into account the corresponding warning error or number.
Summary by File and Line Number	Table consisting of: (filename, linenum, count)	Organizes the errors and warnings by filename and line number.
Summary by File, Line Number, and Message	Table consisting of: (filename, linenum, count, error / warning message)	Expands on the “Summary by File and Line Number” report by including the message that corresponds to the actual error / warning.

Notice the use of the word *Unique* in the “Summary” report. *Unique* errors and warnings are only counted once per line of a file, not multiple times per line. Consider the case of a warning that occurs in a header file that is included in 1000 other source files. If you estimate the effort of porting your application by counting each warning, then you will over-estimate the level of effort required to fix this problem by a factor of 1000. “Total Warnings” would include all 1000 occurrences, but the “Total Unique Warnings” would include only one occurrence for that single line.

Large differences between total error/warning statistics and total *unique* error/warning statistics point to the existence of a problem in one or more header files. This is a good place to start with code cleaning. Locating and highlighting these files is the primary reason why the remaining reports (other than the Summary Report) do not consider *uniqueness*. By counting all occurrences, these reports help you quickly zero in on the files and line numbers where code cleaning should first focus.

Application of the Error Warning Summarization Utility

The value of the Error Warning Summarization Utility becomes apparent when trying to manage the port of a large application to the IA-64 platform. The task of estimating the effort to code clean an application with 10,000 files, for example, and the task of performing the cleaning work can appear daunting. Where do you begin? How do you track progress? Where do you direct resources? The summary reports are designed to provide you this information.

Before the code cleaning begins, try a preliminary build of the application for IA-64 to get an initial estimate of the effort that will be involved. Pass the results of the build through the utility to generate reports. The Summary report gives you top-level information on the total number of errors and warnings that the compiler raises as problems for the application. Recall that *unique* errors and warnings are the best for initial counts, since they indicate the number of lines of code with problems (multiple warnings on the same line are counted only once). In addition, the Summary report indicates how much of the application is able to compile, which is another useful indicator. The Summary by Directory report gives you an idea of what areas of the code are experiencing the most problems. While this is not all that you will need to assess effort for a port, it is the first hard data that you will have available to gain insight into the porting issues for your application.

Once the code cleaning effort has begun, use the Summarization Utility to track progress and to direct resources to the “hot” code cleaning areas that require attention. Use the top-level data in the Summary report as a week-by-week metric for how the process is proceeding. Based on tracking the metrics over time, you can adjust your expected completion dates. For directing the code cleaning effort, use the reports that summarize by directory, by file, and by line number to allocate resources. All these reports can be used to identify hotspots that need code cleaning attention. The process is iterative in the sense that some code cleaning fixes may cause cascading effects to other parts of the code. With each build output after a round of cleaning, these reports will highlight the new areas requiring attention until all issues have been covered.

Finally, once code cleaning is complete, use the Summarization Utility to watch for regressions as your team works to develop for both 32-bit and for 64-bit platforms, using a single source code base. Until 64-bit habits are firmly established in the developers’ day-to-day coding, regressions in the form of errors or warnings during a 64-bit build can be expected to appear. These regressions are immediately identifiable using the reports from the Summarization Utility. By participating as part of the regular build process, the Summarization Utility enables timely identification and resolution of regressions problems.

Conclusion

The Error Warning Summarization Utility provides a variety of Summary reports that you can use at each stage in the process of porting to the IA-64 platform. In the early stages, use the Summary reports to help in assessing the level of effort involved in a port to the 64-bit architecture. During the porting phase, use the Summary reports as metrics to track progress and to guide the allocation of resources to application code sections needing particular attention. After the porting stage, use the utility to watch for regressions as you continue your application's development on both IA-32 and on IA-64 architectures.

Appendix

Installation

The Error Warning Summarization Utility is written in the Python programming language, which is an Open Source programming language. Python is a copyrighted language (http://www.python.org/doc/Copyright.html), but it is freely distributable, even for commercial use. Python is required to run the summarization utility. Since Intel does not distribute Python binaries, here are the instructions for downloading and installing Python yourself:

Download py152.exe from the Python web site (http://www.python.org/download/download_windows.html). This is an InstallShield* executable that installs the core Python interpreter along with supporting modules. Install by executing this image.

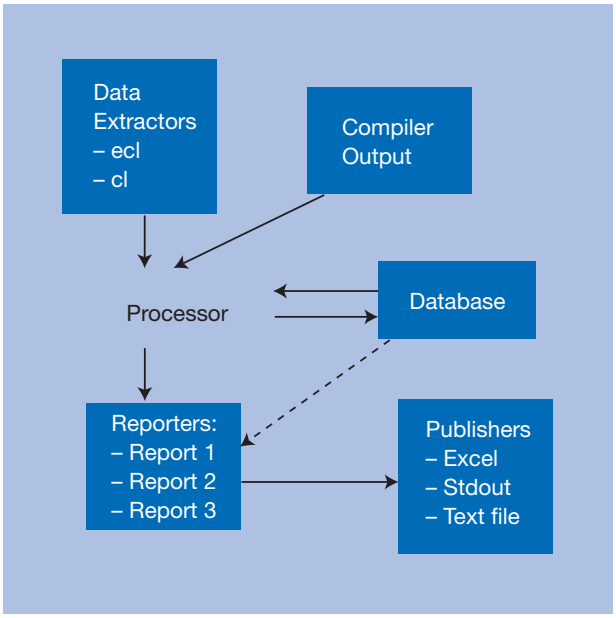
Download win32all-128.exe from Mark Hammond's Python web site (http://starship.python.net/crew/mhammond/win32/Downloads.html). This is an InstallShield package that includes Python extensions specifically targeted at Microsoft Windows platforms. It includes the COM interface that is required if you want to use the `-Excel` option. Install by executing this image.

In the command window that you use to run this utility, add the directory containing python.exe to your path.

Extending the Utility

If you are interested in extending this utility, review the following top-level design description to help you understand more about the utility design and about Python.

The figure below shows the design components of the Summarization Utility. Components include data extractors, a database, reporters, publishers, and a processor. The Data Extractors are responsible for parsing the output from an application build-log file and for extracting the necessary information to be placed in the Database component. A Data Extractor is required for each supported compiler. The Database component is responsible for storing the data and for servicing queries made by the Reporters. The Reporters gather data from the Database component and prepare it for publishing. Publishers take the data from Reporters and publish the information to their respective mediums.



Because of the design of the utility, it is relatively straightforward to add a Data Extractor to support a new compiler, or to add a new Reporter to summarize data, or to add a new Publisher to write reports to a different type of file. It is beyond the scope of this application note to go into more detail on how to extend a particular component. However, the source is available to you for closer examination.

Knowledge of Python and access to the Python source code are requirements for extending this utility. The source code is included with this document. To learn more about Python, review the following information, in order, at http://www.python.org/doc/Intros.html:

- Instant Python
- Getting Started With Python
- Python Tutorial
- Reference: Python Documentation (http://www.python.org/doc)
- Download Python: http://www.python.org/download/download_windows.html